

t8code - modular adaptive mesh refinement in the exascale era

Johannes Holke¹, Johannes Markert¹, David Knapp¹, Lukas Dreyer¹, Sandro Elweijer¹, Niklas Böing¹, Chiara Hergl¹, Prasanna Ponnusamy¹, Jakob Fussbroich¹, Tabea Leistikow¹, Florian Becker¹, Ioannis Lilikakis¹, Julia Weber¹, Sven Goldberg¹, Veli Ünlü¹, and Achim Basermann¹

¹ German Aerospace Center (DLR), Institute for Software Technology, Cologne, Germany
Corresponding author * These authors contributed equally.

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#)
- [Repository](#)
- [Archive](#)

Editor: [Open Journals](#)

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

In this paper, we present our scalable dynamic adaptive mesh refinement (AMR) library t8code, which was officially released in 2022 (Holke et al., 2022). t8code is written in C/C++, open source, and readily available at www.dlr-amr.github.io/t8code. It is developed and maintained at the Institute for Software Technology of the German Aerospace Center (DLR). The software library provides fast and memory efficient parallel algorithms for dynamic AMR to handle tasks such as mesh adaptation, load-balancing, ghost computation, feature search and more. t8code can manage meshes with over one trillion mesh elements (Holke et al., 2021) and scales up to one million parallel processes (Holke, 2018). It is intended to be used as mesh management backend in scientific and engineering simulation codes paving the way towards high-performance applications of the upcoming exascale era.

Statement of Need

Adaptive Mesh Refinement has been established as a successful approach for scientific and engineering simulations over the past decades (Babuvška & Rheinboldt, 1978; Bangerth et al., 2007; Dörfler, 1996; Teunissen & Keppens, 2019). By modifying the mesh resolution locally according to problem specific indicators, the computational power is efficiently concentrated where needed and the overall memory usage is reduced by orders of magnitude. However, managing adaptive meshes and associated data is a very challenging task, especially for parallel codes. Implementing fast and scalable AMR routines generally leads to a large development overhead motivating the need for external mesh management libraries like t8code.

Currently, t8code's AMR routines support a wide range of element types: vertices, lines, quadrilaterals, triangles, hexahedra, tetrahedra, prisms, and pyramids. Additionally, implementation of other refinement patterns and element shapes is possible. See Figure 1 for an exemplary adapted mesh managed by t8code for visualizing earth mantle convection data.

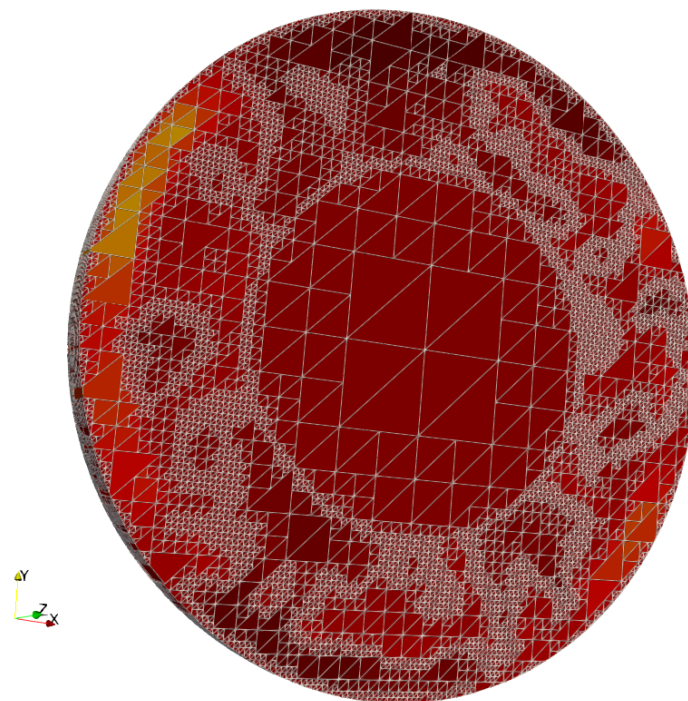


Figure 1: Visualization (2D slice) of an adapted t8code mesh for a visualization of earth mantle convection data.

Fundamental Concepts

t8code is based on the forest-of-trees approach. Starting point for the usage of t8code is an unstructured input mesh, which we denote a coarse mesh. This coarse mesh describes the geometry of the computational domain. Each of the coarse mesh cells are then viewed as the root of a refinement tree. These trees are refined recursively in a structured pattern, resulting in a collection of trees, which we call a forest. t8code stores only a minimal amount of information about the finest elements of the mesh - the leaves of the trees - in order to reconstruct the whole forest.

By enumerating the leaves in a recursive refinement pattern we obtain a space-filling curve (SFC) logic. Via these SFC, all elements in a refinement tree are assigned an index and are stored in the linear order of these indices. Information such as coordinates or element neighbors do not need to be stored explicitly but can be deduced from the index and the appropriate information of the coarse mesh. Fast bitwise and integer-based SFC operations ensure optimal runtimes and diminish the need of memory lookups. Moreover, the SFC is used to distribute the forest mesh across multiple processes, so that each one only stores a unique portion of the forest mesh. See [Figure 2](#).

While being successfully applied to quadrilateral and hexahedral meshes ([Burstedde et al., 2011](#); [Weinzierl, 2019](#)), these SFC techniques are extended by t8code in a modular fashion, such that arbitrary element shapes are supported. We achieve this modularity through a novel decoupling approach that separates high-level (mesh global) algorithms from low-level (element local) implementations. All high-level algorithms can be applied to different implementations of element shapes and refinement patterns. A mix of different element shapes in the same mesh is also supported.



Figure 2: Left: Quad-tree of an exemplary forest mesh consisting of two trees (k0, k1) distributed over three parallel processes p0 to p2. The SFC is represented by a black curve tracing only the finest elements (leaves) of each tree. Right: Sketch of the associated mixed shape mesh refined up to level three. Bottom left: The elements saved by p1 and the associated ghost elements (non process local neighbors).

Performance

t8code supports distributed coarse meshes of arbitrary size and complexity, which we tested for up to 370 million coarse mesh cells (Burstedde & Holke, 2017). Moreover, we conducted various performance studies on the JUQUEEN and the JUWELS supercomputers at the Jülich Supercomputing Center. t8code's ghost and partition routines are exceptionally fast with proper scaling of up to 1.1 trillion mesh elements; see Table 1, (Holke et al., 2021). Furthermore, in a prototype code (Dreyer, 2021) implementing a high-order discontinuous Galerkin method (DG) for advection-diffusion equations on dynamically adaptive hexahedral meshes we observe a 12 times speed-up compared to non-AMR meshes with only an overall 15% runtime contribution of t8code; see Figure 3.

# Process	# Elements	# Elem. / process	Ghost	Partition
49,152	1,099,511,627,776	22,369,621	2.08 s	0.73 s
98,304	1,099,511,627,776	11,184,811	1.43 s	0.33 s

Table 1: Runtimes on JUQUEEN for the ghost layer and partitioning operations for a distributed mesh consisting of 1.1 trillion elements.

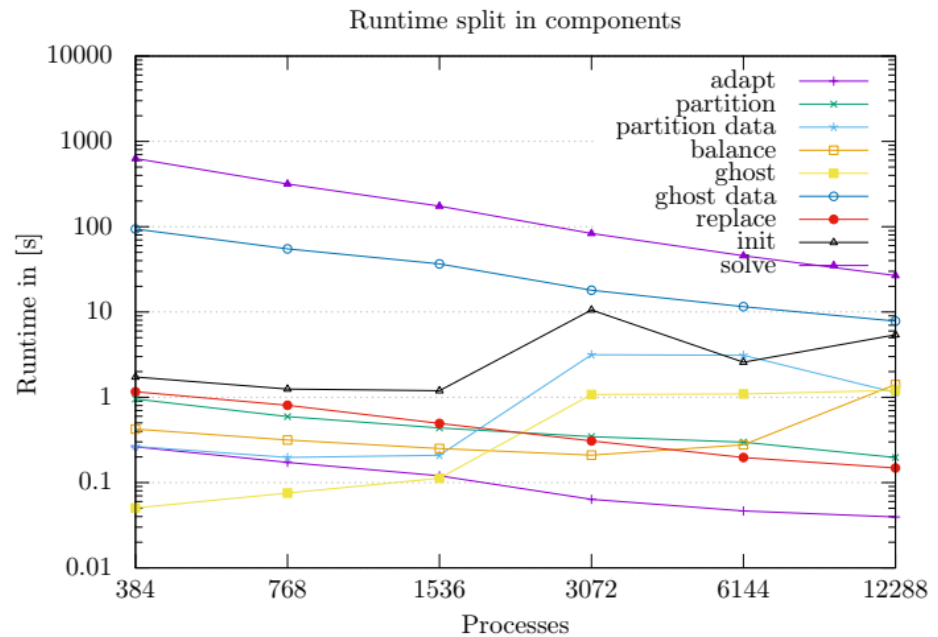


Figure 3: Runtimes on JUQUEEN of the different components of our DG prototype code coupled with t8code. Note that the operations associated with dynamical mesh adaptation (adapt, balance, partition, and ghost) utilize only around 15% of the total runtime largely independent of the number of processes.

Research Projects

Even though t8code is a newcomer to the market, it is already in use as the mesh management backend in various research projects, most notably in the earth system modeling (ESM) community. In the ADAPTEX project t8code is integrated with the Trixi framework (Schlottke-Lakemper et al., 2020) - a modern computational fluid dynamics code written in Julia. Over the next years several ESM applications are planned to couple to this combination, including MESSy, MPTrac, and SERGHEI. Moreover, t8code also plays an important role in several DLR funded research projects, e.g., VisPlore (massive data visualization), HYTAZER (hydrogen tank certification), Greenstars (additive rocket engine manufacturing) and PADME-AM (simulation assisted additive manufacturing).

Further Information

For further information beyond this short note and also for code examples, we refer to our Documentation and Wiki reachable via our homepage www.dlr-amr.github.io/t8code and our technical publications on t8code (Becker, 2021; Burstedde & Holke, 2016, 2017; Dreyer, 2021; Elswijker, 2021; Fußbroich, 2023; Holke, 2018; Holke et al., 2021, 2022; Knapp, 2020; Lilikakis, 2022).

Acknowledgements

Johannes Holke thanks the Bonn International School Graduate School of Mathematics (BIGS) for funding the initial development of t8code. Further development work was funded by the German Research Foundation as part of project 467255783, the European Union via NextGenerationEU and the German Federal Ministry of Research and Education (BMBF) as part of the ADAPTEX and PADME-AM projects. Development work was performed as part of the Helmholtz School for Data Science in Life, Earth and Energy (HDS-LEE) and received

90 funding from the Helmholtz Association of German Research Centres. The development team
 91 of t8code thanks the Institute of Software Technology and the German Aerospace Center
 92 (DLR).

93 The authors state that there are no conflicts of interest.

94 References

- 95 Babuška, I., & Rheinboldt, W. C. (1978). Error estimates for adaptive finite element
 96 computations. *SIAM Journal on Numerical Analysis*, 15(4), 736–754. <https://doi.org/10.1137/0715049>
 97
- 98 Bangerth, W., Hartmann, R., & Kanschat, G. (2007). Deal.II—a general-purpose object-
 99 oriented finite element library. *ACM Trans. Math. Softw.*, 33(4), 24–es. <https://doi.org/10.1145/1268776.1268779>
 100
- 101 Becker, F. (2021). *Removing hanging faces from tree-based adaptive meshes for numerical*
 102 *simulations* [Master's thesis]. Universität zu Köln.
- 103 Burstedde, C., & Holke, J. (2016). A tetrahedral space-filling curve for nonconforming adaptive
 104 meshes. *SIAM Journal on Scientific Computing*, 38, C471–C503. <https://doi.org/10.1137/15M1040049>
 105
- 106 Burstedde, C., & Holke, J. (2017). Coarse Mesh Partitioning for Tree-Based AMR. *SIAM Jour-*
 107 *nal on Scientific Computing*, Vol. 39, C364–C392. <https://doi.org/10.1137/16M1103518>
- 108 Burstedde, C., Wilcox, L. C., & Ghattas, O. (2011). p4est: Scalable Algorithms for Parallel
 109 Adaptive Mesh Refinement on Forests of Octrees. *SIAM Journal on Scientific Computing*,
 110 33(3), 1103–1133. <https://doi.org/10.1137/100791634>
- 111 Dörfler, W. (1996). A convergent adaptive algorithm for poisson's equation. *SIAM Journal on*
 112 *Numerical Analysis*, 33(3), 1106–1124. <https://doi.org/10.1137/0733054>
- 113 Dreyer, L. (2021). *The local discontinuous galerkin method for the advection-diffusion*
 114 *equation on adaptive meshes* [Master's thesis, Rheinische Friedrich-Wilhelms-Universität
 115 Bonn]. <https://elib.dlr.de/143969/>
- 116 Elswijker, S. (2021). *Curved Domain Adaptive Mesh Refinement with Hexahedra*. Hochschule
 117 Bonn-Rhein-Sieg. <https://elib.dlr.de/143537/>
- 118 Fußbroich, J. (2023). *Towards high-order, hybrid adaptive mesh refinement: Implementa-*
 119 *tion and evaluation of curved unstructured mesh elements* [Master's thesis, Technische
 120 Hochschule Köln]. <https://elib.dlr.de/200442/>
- 121 Holke, J. (2018). *Scalable algorithms for parallel tree-based adaptive mesh refinement with*
 122 *general element types* [{PhD} thesis]. Rheinische Friedrich-Wilhelms-Universität Bonn.
- 123 Holke, J., Burstedde, C., Knapp, D., Dreyer, L., Elswijker, S., Uenlue, V., Markert, J., Lilikakis,
 124 I., & Boeing, N. (2022). t8code (Version 1.0.0). <https://doi.org/10.5281/zenodo.7034838>
- 125 Holke, J., Knapp, D., & Burstedde, C. (2021). An Optimized, Parallel Computation of the
 126 Ghost Layer for Adaptive Hybrid Forest Meshes. *SIAM Journal on Scientific Computing*,
 127 C359–C385. <https://doi.org/10.1137/20M1383033>
- 128 Knapp, D. (2020). *A space-filling curve for pyramidal adaptive mesh refinement* [Master's
 129 Thesis]. Rheinische Friedrich-Wilhelms-Universität Bonn.
- 130 Lilikakis, I. (2022). *Algorithms for tree-based adaptive meshes with incomplete trees* [Master's
 131 thesis, Universität zu Köln]. <https://elib.dlr.de/191968/>

- 132 Schlottke-Lakemper, M., Gassner, G. J., Ranocha, H., & Winters, A. R. (2020). *Trixi.jl:*
133 *Adaptive high-order numerical simulations of hyperbolic PDEs in Julia*. <https://github.com/trixi-framework/Trixi.jl>. <https://doi.org/10.5281/zenodo.3996439>
134
- 135 Teunissen, J., & Keppens, R. (2019). A geometric multigrid library for quadtree/octree
136 AMR grids coupled to MPI-AMRVAC. *Computer Physics Communications*, 245, 106866.
137 <https://doi.org/https://doi.org/10.1016/j.cpc.2019.106866>
- 138 Weinzierl, T. (2019). The Peano Software-Parallel, Automaton-based, Dynamically Adaptive
139 Grid Traversals. *ACM Transactions on Mathematical Software*, 45(2), 1–41. <https://doi.org/10.1145/3319797>
140

DRAFT